DOG: an Ontology-Powered OSGi Domotic Gateway

Dario Bonino, Emiliano Castellina, Fulvio Corno

Politecnico di Torino
e-Lite Research Group
Torino - Italy

March 25, 2009
Outline

1. Introduction
2. Objectives
3. DOG
   - Ring 0
   - Ring 1
   - Ring 2
   - Ring 3 bundles
4. DogOnt
5. Semantics in DOG
   - Property verification
6. Conclusions
Information technology in the home

- Remote lighting and appliance control have been used for years (see X10)
- Nowadays domotics is another term for the digital home, including: the networks and devices that add comfort and convenience as well as security;
- Domotics means controlling heating, air conditioning, food preparation, TVs, stereos, lights, appliances and security system of the home
Domotics – Drawbacks (1/2)

Many vendors on the market with not compatible solutions

- Different technologies (bus, powerline, wireless)
- Different protocols (KNX, MyOpen, X10, LonWorks)
- Different device features
- Different sophistication of device firmware (from simple relay to full software-based operation)
Rooted on Simple Electric Automation

- Only simple automation is supported
  - Simple scenarios
  - Fixed, programmed behaviors
  - Simple comfort, security and energy saving policies

- No support for more complex interactions
  - Adaptation to user preferences
  - Context detection
  - Structural verification
  - Static and dynamic reasoning on the house state
DOG: an Ontology-Powered OSGi Domotic Gateway

Outline

1. Introduction
2. Objectives
3. DOG
   - Ring 0
   - Ring 1
   - Ring 2
   - Ring 3 bundles
4. DogOnt
5. Semantics in DOG
   - Property verification
6. Conclusions
Goal

Evolving into Intelligent Domotic Environments (IDEs)

Supporting Interoperation, Integration and Intelligence by

- Adding a single (cheap) device for
  - interoperating different domotic plants
  - implementing complex behaviors
- Modeling environments in a semantic-rich, technology independent way
- Providing suitable querying and reasoning mechanism over the environment model
Domotic Systems vs Smart Home

**Smart Home**
- **Pros**
  - supports complex and intelligent behaviors
- **Cons**
  - home pervaded by sensors and actuators
  - dedicated hardware and software
  - Experimental and futuristic connotation
  - Very expensive

**Domotic Systems**
- **Pros**
  - Commercial solution
  - Modular and (relatively) easy to install and configure
  - Affordable costs
- **Cons**
  - Sparse technologies
  - Only supports simple automation
  - No support for intelligent behaviors
Starting considerations

- The sparseness of domotics solutions, the differences in languages, communication means and protocols is very similar to the “old web”

- Semantic Web technologies can help solving
  - Interoperation issues
  - Integration of different technologies

- and can support home intelligence through
  - Reasoning
  - Context Modeling
  - ...
Anatomy of an Intelligent Domotic Environment

DOG: an Ontology-Powered OSGi Domotic Gateway
1 Introduction
2 Objectives
3 DOG
   - Ring 0
   - Ring 1
   - Ring 2
   - Ring 3 bundles
4 DogOnt
5 Semantics in DOG
   - Property verification
6 Conclusions
DOG (Domotic OSGi Gateway) is a Domotic House Gateway designed for transforming commercial Domotic Systems into Intelligent Domotic Environments.

Based on OSGi architecture.

DOG provides

- Interoperation between different domotic networks through proper drivers
- Technology independent, ontology-based, house and device modeling
- Advanced, inter-network, rule-based scenario definition and operation

DogOnt is the ontology model lying at the basis of DOG
DOG: an Ontology-Powered OSGi Domotic Gateway

DOG Architecture

- **Ring 0**: the DOG common library and communication between the OSGi platform and the other rings
- **Ring 1**: interface to the various domotic networks
- **Ring 2**: routing infrastructure for messages and intelligence core of DOG (DogOnt)
- **Ring 3**: access to external applications
Outline

1. Introduction
2. Objectives
3. **DOG**
   - Ring 0
   - Ring 1
   - Ring 2
   - Ring 3 bundles
4. **DogOnt**
5. **Semantics in DOG**
   - Property verification
6. Conclusions

DOG: an Ontology-Powered OSGi Domotic Gateway
Ring 0 bundles

**Dog library**
Library repository for all other DOG bundles. Provides the interfaces of the services implemented by DOG bundles.

**Platform Manager**
Handles the start-up of the whole system and manage the life cycle of DOG bundles.

**Configuration Registry**
Stores configuration information about each bundle.
Outline

1. Introduction
2. Objectives
3. DOG
   - Ring 0
   - Ring 1
   - Ring 2
   - Ring 3 bundles
4. DogOnt
5. Semantics in DOG
   - Property verification
6. Conclusions
Ring 1 bundles

Network Drivers

- A Network Driver for each different domotic technology (e.g. KNX, OpenWebNet, X10, etc.)
- Self-configuration phase in which they retrieve the list of devices from the House Model.
- Network drivers translate messages back and forth between Dog bundles and network-level gateways.
Ring 2 bundles (1/2)

**Message Dispatcher**
Internal router, delivering messages (commands, state polls or notifications) to the correct destinations.

**Executor**
Semantically validates the command received from the API and forwards to the Message Dispatcher.
Ring 2 bundles (2/2)

**Status**
Caches the states of all devices controlled by DOG.

**House Model**
Intelligence core of DOG. Based on DogOnt ontology.
Outline

1 Introduction
2 Objectives
3 DOG
   - Ring 0
   - Ring 1
   - Ring 2
   - Ring 3 bundles
4 DogOnt
5 Semantics in DOG
   - Property verification
6 Conclusions
Ring 3 bundles

**API**
Retrieve the house configuration, to send commands to devices and to receive house events.

**XmlRPC bundle**
It provides an XML-RPC endpoint for services offered by API bundle. It enables non-OSGi applications to control DOG.
Outline

1. Introduction
2. Objectives
3. DOG
   - Ring 0
   - Ring 1
   - Ring 2
   - Ring 3 bundles
4. DogOnt
5. Semantics in DOG
   - Property verification
6. Conclusions

DOG: an Ontology-Powered OSGi Domotic Gateway
DogOnt is an ontology model designed for supporting Interoperation, Integration and Intelligence in domotic environments

- Building Thing
- Building Environment
- State
- Functionality
- Domotic Network Component
BuildingThing

- Models all the elements of a Building Environment divided into
  - Controllable
  - UnControllable
- The UnControllable sub-tree allows to model
  - Furniture elements
  - Walls, floors, ceilings and other architectural elements (Architectural sub-tree)
Environment Modeling (2/2)

Building Environment

- Models rooms and architectural spaces composing a house
  - Rooms
  - External spaces such as garages, garden, etc.
Devices are modeled independently from specific technologies

3 Modeling axes:

- **Typology** - describes the type of device, separating appliances and devices belonging to house plants
- **Functionality** - describes the tasks that a device can accomplish, by defining the available commands
- **State** - describes the conditions in which a device can be (e.g. a Lamp can be ON or OFF)

Technology specific aspects are modeled through separate classes

- **NetworkComponent** - the root concept for modeling every network specific information, its sub-classes reflect the different networks supported by DOG.
Typology

Controllable devices taxonomy

- Appliances
  - Brown Goods (TV, HiFi,...)
  - White Goods (Fridge, Dishwasher,...)
- HousePlants
  - Electric
  - HVAC (Heating Ventilation & Air Conditioning)
  - Security
**Control Functionalities**
- Model the ability of a device to be controlled
- Define the possible commands and their range (needed for continuous functionalities)
- Almost every Controllable has a control functionality

**Notification Functionalities**
- Model the ability of a device to issue a notification about state/configuration changes
- Define the possible notifications
- Typical of Sensors and Buttons/Switches

**Query Functionalities**
- Model the ability of a device to be queried about its state/configuration
- It’s defined *for all* Controllables
Functionalities (2/3)

Every Functionality class is subdivided into

- **Continuous Functionalities**
  - Model the ability to change device properties in a continuous manner (e.g., dimming the light emitted by a lamp)

- **Discrete Functionalities**
  - Model the ability to abruptly change device properties (e.g., switching a lamp On)
Functionalities (3/3)
States are classified according to the kind of values they can assume

- **Continuous states**
  - Model continuously changing qualities (e.g. the current dimming level of a lamp)
  - The current state value is stored in the `continuousValue` property.

- **Discrete states**
  - Model discretely changing qualities (e.g. the lamp being On or Off)
  - The current state value is stored in the `discreteValue` property.
  - Possible states are listed in the `possibleStates` property.
States (2/2)

- State
  - Continuous State
    - LightIntensity State
      - LightIntensityState Instance
        - sample_dimmer_lamp
  - valueContinuous

DOG: an Ontology-Powered OSGi Domotic Gateway
DimmerLamp modeling example

- sample_living_room
- switch_sample_dimmer_lamp
- valueContinuous
- 100%
- LightIntensityState Instance
- hasState
- inherited from Lamp
- OnOffFunctionality Instance
- commands
- On
- Off
- LightRegulation FunctionalityInstance
- MaxValue 100%
- MinValue 0%
- commands
- StepUp
- StepDown
- Set

queryFunctionality Instance (getInstance)

commands Get

hasFunctionality

inherited from Controllable
Outline

1. Introduction
2. Objectives
3. DOG
   - Ring 0
   - Ring 1
   - Ring 2
   - Ring 3 bundles
4. DogOnt
5. Semantics in DOG
   - Property verification
6. Conclusions

DOG: an Ontology-Powered OSGi Domotic Gateway
Experimental set-up

Technologies
- Eclipse Equinox OSGi framework
- Jena and Pellet
- MyOpen and KNX

Components
- DOG runs on an ASUS eeePC701
  - 900MHz Celeron processor
  - 512MByte RAM
  - 4GByte SSD
- KNX demo case built by the authors
- MyOpen demo case offered by BTicino
Reference Environment

Domotic Devices

- 27 Push Buttons
- 7 Lamps
- 23 Plugs
- 7 Door Actuators
- 7 Door Sensors
- 6 Window Actuators
- 6 Window Sensors
- 6 Shutter Actuators
- 5 Infrared Sensors
- 6 Smoke Sensors

DOG: an Ontology-Powered OSGi Domotic Gateway
The House Model is the core of the DOG intelligence.
It is based on a formal model defined by DogOnt ontology.
DogOnt is designed for supporting Interoperation, Integration and Intelligence in domotic environments.
DogOnt supports several critical features of DOG.
A central **point of configuration** for devices

- **specific uniform set of devices, states and functionalities**

- Enables **syntactic and semantic check of commands**

- Top-down **inter-plant** scenarios which involve devices

- Provides interoperation between plants (e.g. allowing a BTicino button to control a KNX light)
Ontology-based Operations in DOG

Operations supported by DogOnt

- Installation (∼ 40s)
  - Model Reasoning (transitive closure + classification)
- Start-up (< 3s)
  - SPARQL queries for associating devices to drivers
- Validation (<100ms)
  - SPARQL queries for gathering allowed commands and their ranges
  - Comparison between requested and allowed operations
- Inter-network scenarios
  - SPARQL queries for gathering specific device types (e.g. Lamps)
  - Generation of commands on the basis of device types (e.g. all Lamps ON)
Start-up

Start-up process:

1. Platform Manager
   - start()
2. Network Driver (Konnex)
   - start()
   - getAllDevices(Konnex)
   - Konnex devices
   - * repeat until all devices are mapped
     - Create Mappings
     - Generalize(?x)
     - Generalize(?x)
     - Father(?x)

3. Dispatcher
   - Konnex devices
4. House Model
   - Konnex devices
   - getAllDevices(Konnex)
Command Validation

Flowchart:

1. Start Validation
2. Syntax OK?
   - Yes: get Device Description
   - No: SYNTACTIC VALIDATION
3. get Device Description
4. commands AND values match?
   - Yes: Check Semantics
   - No: Discard message
5. Check Syntax
6. get Device Description
7. return Description
8. Check Semantics
9. Dog Message
10. Forward message

API -> Executor -> House Model -> Dispatcher
Controllable query excerpt

SELECT DISTINCT ?x ?y WHERE {
  ...
  UNION
  { ?x rdfs:subClassOf dogont:Controllable . ?x rdfs:subClassOf ?s.
    ?s rdfs:subClassOf [rdf:type owl:Restriction;
      owl:onProperty dogont:hasFunctionality;
      owl:someValuesFrom ?y] .
    ?y rdfs:subClassOf dogont:Functionality; }
  UNION
  { ?x rdfs:subClassOf dogont:Controllable . ?x rdfs:subClassOf ?s.
    ?s rdfs:subClassOf [rdf:type owl:Restriction;
      owl:onProperty dogont:hasFunctionality;
      owl:allValuesFrom ?u] .
    ?y rdfs:subClassOf dogont:Functionality; }
  ...
} . FILTER(?x != owl:Nothing) . FILTER(?x != owl:Thing)
} ORDER BY ?x ?y
Inter-network scenarios

External App. | API | House Model | Executor | Dispatcher
---|---|---|---|---
allLampsOFF | ?lamp | all Lamps | Transitive closure | 2

Set(?x,OFF) | * for all Lamp(?x) | Validate | Set(?x,OFF) | Set(?x1, OFF) | Set(?x2, OFF)

To Network Drivers | 1

DOG: an Ontology-Powered OSGi Domotic Gateway
Advanced Intelligence in DOG

- Transitive closure and Classification Reasoning to **decouple evolution of the model and domotic systems**
- Structural verification of domotic environments through SWRL constraints
- Dynamic detection of safety critical situations (smoke propagation, safe exit path) using rule-based reasoning
- Automatic generation of interoperation rules from DogOnt
Outline

1. Introduction
2. Objectives
3. DOG
   - Ring 0
   - Ring 1
   - Ring 2
   - Ring 3 bundles
4. DogOnt
5. Semantics in DOG
   - Property verification
6. Conclusions
Property types

- **type perspective**
  - **structural** properties: environment properties related to design, usability and normative issues; usually checked off-line
  - **state** properties: current, or the possible, value(s) of specific environment characteristics, given the environment structure and the current state of the installed devices.

- **inference perspective:**
  - **Direct** properties, that can be easily checked by applying direct, forward rules.
  - **Recursive** properties, where the results of direct inferences are recursively propagated to the whole environment.
  - **Multi-stage** properties, which require the sequential execution of different rule bases.
# Structural Properties

<table>
<thead>
<tr>
<th>Domain</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architectural, accessible Controls normative</td>
<td></td>
<td>every button, switch or control must be positioned between 40 and 120 cm from the floor.</td>
</tr>
<tr>
<td>Operational</td>
<td>usableLamp</td>
<td>all lamps in a <em>Room</em> must be controllable by a switch in the same or in an adjacent <em>Room</em>.</td>
</tr>
<tr>
<td>Architectural reachability</td>
<td></td>
<td>Every <em>Room</em> shall be reachable from any other <em>Room</em> in a finite number of room traversals.</td>
</tr>
</tbody>
</table>
## State Properties

<table>
<thead>
<tr>
<th>Domain</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operational</td>
<td>roomIllumination</td>
<td>A <em>Room</em> is illuminated if at least a <em>Lamp</em> located inside the same <em>Room</em> is lit or if there is an active lamp in an adjacent <em>Room</em>.</td>
</tr>
<tr>
<td>Comfort</td>
<td>mosquitoFree</td>
<td>Given the current <em>State of Doors</em> and <em>Windows</em>, label all <em>Rooms</em> that can be reached by mosquitoes as <em>not-mosquito-free</em> and all the remaining rooms as <em>mosquito-free</em>.</td>
</tr>
<tr>
<td>Safety</td>
<td>smokeFree</td>
<td>Given the current <em>State of Smoke-Sensors</em>, <em>Doors</em> and <em>Windows</em>, evaluate where smoke can propagate, and label <em>Rooms</em> reachable by smoke as <em>not-smoke-free</em> and rooms isolated from smoke as <em>smoke-free</em>.</td>
</tr>
</tbody>
</table>
## Cross-classification of properties

<table>
<thead>
<tr>
<th></th>
<th>Direct</th>
<th>Recursive</th>
<th>Multi-stage</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Structural</strong></td>
<td>accessibleControls usableLamp</td>
<td>reachability</td>
<td></td>
</tr>
<tr>
<td><strong>State</strong></td>
<td>roomIllumination*</td>
<td>mosquitoFree smokeFree*</td>
<td>validEscape</td>
</tr>
</tbody>
</table>
usableLamp: formalization

\[ \text{Lamp}(x) \land \text{Control}(y) \land \text{Room}(r) \land \text{Room}(r1) \land \]
\[ \text{hasControl}(x,y) \land ((\text{isIn}(x,r) \land \text{isIn}(y,r)) \lor \]
\[ (\text{isIn}(x,r) \land \text{isIn}(y,r1) \land \text{adjacentTo}(r,r1))) \]
\[ \implies \text{usableLamp}(x) \]
usableLamp: SWRL implementation

Lamp(?x)^Control(?y)^Room(?r)^
hasControl(?x,?y)^isIn(?x,?r)^
isIn(?y,?r)
-> isUsable(?x,true^^xsd:boolean)

Lamp(?x)^Control(?y)^Room(?r)^
Room(?r1)^hasControl(?x,?y)^
isIn(?x,?r)^isIn(?y,?r1)^
WallOpening(?wo)^Wall(?w)^
hasWallOpening(?w,?wo)^
hasWall(?r,?w)^hasWall(?r1,?w)
-> isUsable(?x,true^^xsd:boolean)
Outline

1. Introduction
2. Objectives
3. DOG
   - Ring 0
   - Ring 1
   - Ring 2
   - Ring 3 bundles
4. DogOnt
5. Semantics in DOG
   - Property verification
6. Conclusions
Conclusions

- We developed DOG: an ontology-powered OSGi Domotic Gateway
- Dog currently uses DogOnt ontology, that allows to control several, different, domotic plants, at the same time
- Dog will transform your Domotic plants into Intelligent Domotic Environments.

http://domoticdog.sourceforge.net
References

- BONINO D; CASTELLINA E; CORNO F.; GALE A; GARBO A; PURDY K; SHI F, A blueprint for integrated eye-controlled environments, UNIVERSAL ACCESS IN THE INFORMATION SOCIETY, 2009, Vol. 8/4, ISSN: 1615-5289, DOI: 10.1007/s10209-009-0145-4


This document is licensed under the Creative Commons Attribution-Noncommercial-Share Alike 3.0 Unported License.

http://creativecommons.org/licenses/by-nc-sa/3.0/